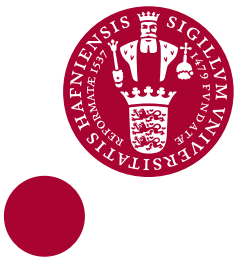
UNIVERSITY OF COPENHAGEN



Deep learning as a parton shower

Monk, J. W.

Published in:
Journal of High Energy Physics

DOI:
10.1007/JHEP12(2018)021

Publication date:
2018

Document version
Publisher's PDF, also known as Version of record

Citation for published version (APA):
Monk, J. W. (2018). Deep learning as a parton shower. *Journal of High Energy Physics*, *2018*(12), [021]. https://doi.org/10.1007/JHEP12(2018)021

Download date: 04. maj. 2023

Published for SISSA by Springer

Received: *August 3, 2018*
Revised: *October 9, 2018*
Accepted: *November 12, 2018*
Published: *December 5, 2018*

JHEP12(2018)021

Deep learning as a parton shower

J.W. Monk

Niels Bohr Institute, University of Copenhagen, Blegdamsvej, Copenhagen, Denmark

E-mail: jmonk@cern.ch

Abstract: We make the connection between certain deep learning architectures and the renormalisation group explicit in the context of QCD by using a deep learning network to construct a toy parton shower model. The model aims to describe proton-proton collisions at the Large Hadron Collider. A convolutional autoencoder learns a set of kernels that efficiently encode the behaviour of fully showered QCD collision events. The network is structured recursively so as to ensure self-similarity, and the number of trained network parameters is low. Randomness is introduced via a novel custom masking layer, which also preserves existing parton splittings by using layer-skipping connections. By applying a shower merging procedure, the network can be evaluated on unshowered events produced by a matrix element calculation. The trained network behaves as a parton shower that qualitatively reproduces jet-based observables.

Keywords: Phenomenological Models, Jets

ArXiv ePrint: 1807.03685

Open Access, © The Authors.
Article funded by SCOAP[3].

https://doi.org/10.1007/JHEP12(2018)021

Contents

JHEP12(2018)021

1 Introduction

The renormalisation group provides a set of rules that describe how a system evolves under a re-scaling transformation. This is expressed in parton shower models by the repeated evaluation of a splitting kernel over an ordered hierarchy of scales, which results in the self-similarity that is a characteristic of renormalisable models. We have previously exploited this behaviour by using wavelet decomposition [1] to extract features from radiation patterns in proton-proton collision events at both small and large angles.

A single layer in a convolutional neural network (CNN) is very similar to a single level wavelet decomposition. Indeed, with appropriate network parameters, the CNN *is* a wavelet decomposition. Since the wavelet basis can be used to reveal the angular evolution of a parton shower, this raises the intriguing possibility that a CNN could be structured in such a way that it encodes, and behaves as, a parton shower model. That the hierarchical structure of deep learning architectures is formally connected to the behaviour of the renormalisation group is an area of active interest, see e.g. [2–4]; in this paper we will make the connection between these ideas obvious by constructing a toy parton shower model using a deep learning neural network whose design has been inspired by wavelet decomposition.

An autoencoding neural network takes an input of high dimensionality, compresses it to a bottleneck of a small number of network nodes, then reinflates the compressed values

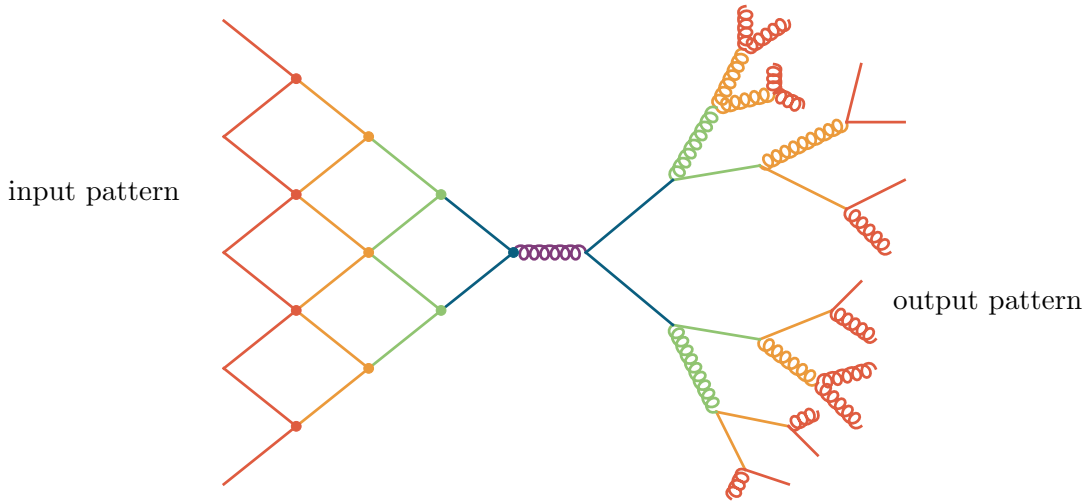


Figure 1. Comparison between the structure of a convolutional autoencoder and a parton shower. A pattern is input on the left and the value at each network node is determined by the weighted sum of the connected nodes.

to recover the input data as the target. In so-doing, the behaviour of the input data is encoded in the network parameters. The compression stage of a convolutional autoencoder uses a series of convolutional layers interspersed with pooling layers to repeatedly reduce the dimensionality of the input. Having compressed the data at the bottleneck, the re-inflation half of the autoencoder again uses (different) convolutional layers interspersed with up-scaling. Figure 1 shows the general form of a convolutional autoencoder, where the use of gluon lines is intended to make explicit the similarity between the re-inflation stage and an iterative parton shower.

The scaling behaviour of a parton shower means that the (de-) convolutional kernels used in each layer of the autoencoding CNN should be related to those used in all the other layers. While effects like colour coherence or the divergence of the strong coupling at low energy mean that QCD is not exactly scale invariant, parton showers are used in an energy regime where scale invariance is a good approximation. In practice, this means that the same convolutional kernels can be used in each layer, which ensures self-similarity over the different angular scales that the network layers represent. Despite the re-use of the same kernels at each scale, the running of the strong coupling can be approximated by evolving the relative contributions of the different available kernels. The re-use of the same kernel in multiple layers also means there are a relatively small number of independent network parameters, even if the network is deep. This multi-scale coarse-graining approach means that behaviour that the network learns at one angular scale is applied to all angular scales down to the cut-off.

Other results have also noted that the self-similarity of parton showers can be encoded in a recursive neural network. For example, the JUNIPR model described in [5] uses a recursive network based on hierarchical $1 \to 2$ particle splittings to learn probability distributions for particle emissions within jets. Conversely, image-based network designs such as [6] have also been used to generate jet images directly. However, as far as we are

JHEP12(2018)021

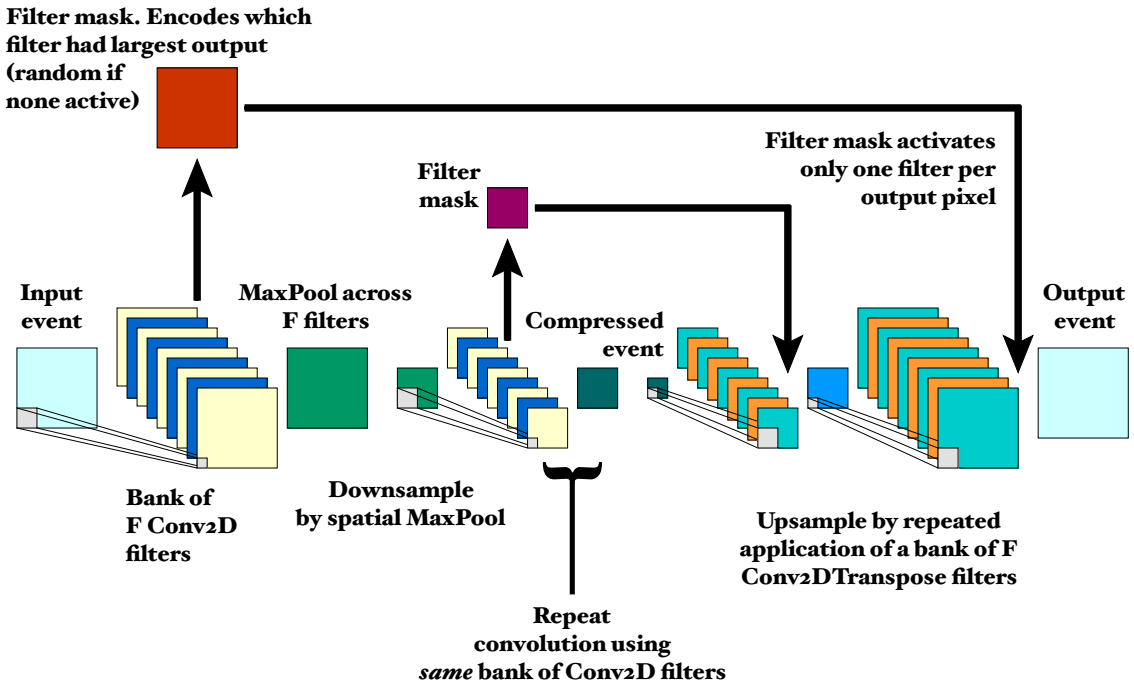


Figure 2. Overview of the autoencoding CNN. An input event image enters the network on the left and is repeatedly processed by a bank of F `Conv2D` filters. Having been completely compressed, the event is reinflated using `Conv2DTranspose` layers together with a special `FilterMask` layer.

aware, our model is the first that combines both recursion with an image-based approach that can generate entire events and that can be merged with a fixed order matrix element calculation. A more detailed discussion of the similarities and differences between the three approaches will be presented in section 6.

Building a deep learning network that can approximate the behaviour of QCD is a useful exercise for several reasons: such a network can help provide insights into *why* neural networks (sometimes!) work so well for analysis tasks; the network can extract features and observables directly from data, which can be used to confront existing shower models; the evolution of the network parameters with depth in the network can provide some insight into the structure of showers; the trained autoencoder will not fit data that is different from that it is trained on, hence could be used to identify signal that differs from the QCD backgound; and the toy model trained directly on data can provide a useful comparison to existing methods for tuning Monte Carlo models.

2 Design of the autoencoding CNN

2.1 Network structure

The layout of the autoencoding CNN is shown in figure 2. A set of F 2-dimensional convolutional layers (`Conv2D`) is used in the compression half of the autoencoder, and F transposed 2-dimensional convolution layers (`Conv2DTranspose`) are used in the reinflation

JHEP12(2018)021

half of the autoencoder. The `Conv2D` layers are defined with kernels of size $k \times k$ and pad the input so that the output from the layer has the same size as the input. The `Conv2DTranspose` layers are also defined with kernels of size $k \times k$, but use a stride size of k together with padding of the input to ensure that the output from the layer has dimensions $kN \times kN$, where the dimensionality of the input to the layer is $N \times N$. Event data is provided to the autoencoding CNN model in the form of pixel arrays that represent the emissions of energy from each proton-proton collision.

The input image is passed through the F `Conv2D` layers, which results in a stack of F output images, each of which is the same size as the original input image. A max-pooling layer is used across these F images so that each pixel site output by the max-pooling layer is the maximum value of the corresponding pixel sites in the F input images. The output of the max-pooling across the F convolutions is thus a single image the same size as the initial input image. This single image is then downsampled by a further spatial max-pooling layer. The spatial max-pooling uses a pool size of k, meaning that an initial image of size $kN \times kN$ is downsampled to a $N \times N$ image. The combined effect of the filter max-pooling followed by spatial max-pooling is to combine a $k \times k$ region of pixels into a single pixel, using the filter that best matches the shape of the input for that region.

This $N \times N$ image is once again passed to the *same* set of F `Conv2D` layers, followed by the max-pooling layers, to further reduce the dimensionality of the image. The sequence of convolution followed by max-pooling is repeated until the output image size is $k \times k$. Note that if such a $k \times k$ image were again to be passed through the `Conv2D` and max-pooling layers, the result would be a single pixel.

The fully compressed $k \times k$ image is then passed to the set of F `Conv2DTranspose` layers, which results in a stack of F $k^2 \times k^2$ images that can be written as a single tensor T_{ijk}, where index i is in the range $\{0 \ldots F\}$ and j and k are both in the range $\{0 \ldots k^2\}$. This stack of up-scaled images is converted to a single image by using a custom layer that we have named `FilterMask`. The `FilterMask` layer, M_{ijk}, uses the corresponding $k^2 \times k^2 \to k \times k$ downsampling in the compression stage of the autoencoder to decide which of the pixels in T_{ijk} should be used. `FilterMask` is a tensor with the same $F \times k^2 \times k^2$ shape as T_{ijk}, but in which each pixel is either zero or one, as described in equation (2.1)

$$M_{ijk} = \begin{cases} 1 & \forall m, C_{ijk} \geq C_{mjk} \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

where C_{ijk} is the output of the corresponding compression filters on the compression stage of the network. Pixels are one if the corresponding pixel in the `Conv2D` output image is the maximum in that stack of pixels. All other pixels in the `FilterMask` are zero. The output stack of images from the `Conv2DTranspose` is multiplied by the mask M_{ijk}, so that at most one pixel in the stack is non-zero. This set of masked images is then converted to a single image, S_{jk}, by summing all of the pixels in a stack as given in equation (2.2)

$$S_{jk} = \sum_{i=0}^{F} M_{ijk} T_{ijk} \,. \tag{2.2}$$

JHEP12(2018)021

The `FilterMask` transfers information about which `Conv2D` filter was active in the compression stage to the reinflation stage, meaning that the `Conv2DTranspose` filter kernels are dependent on the upstream `Conv2D` kernels. This also means that there is a mechanism by which a splitting present in the input event can be preserved through the network, while admitting a degree of randomness. The `FilterMask` keeps a counter of the number of times each `Conv2D` filter was active during training, and converts these rates into a set of probabilities that have unit sum. In the case that all `Conv2D` filters produce identical (therefore zero) output in the same pixel, the `FilterMask` randomly picks a single filter to activate according to its recorded probability. This means that if a single active pixel is passed into the CNN, it will be propagated to the $k \times k$ bottleneck as a single pixel but then reinflated using randomly activated `Conv2DTranspose` upscaling filters. It is this feature of the `FilterMask` - preserving input splittings when they exist, while producing random splittings when none are present — that allows the autoencoding CNN to behave as a parton shower.

Having inflated the $k \times k$ image to $k^2 \times k^2$ and applied the `FilterMask` derived from the $k^2 \times k^2 \to k \times k$ compression stage, the output $k^2 \times k^2$ image is once again passed to the stack of `Conv2DTranspose` filters, which output a stack of $k^3 \times k^3$ images. A `FilterMask` is once again used to merge the stack, but this time the mask is taken from the $k^3 \times k^3 \to k^2 \times k^2$ compression stage. Different `FilterMask` layers, with different learned filter probabilities, are thus used at the different compression and reinflation levels. This allows the filter activation probabilities to evolve with angular scale.

The process of applying the stack of `Conv2DTranspose` filters, followed by merging with the corresponding `FilterMask` from the compression stage, is repeated until the image is the same size as the original input image.

The CNN is implemented in python using Keras [7] with the TensorFlow [8] backend and is evaluated using a pair of Nvidia 1080 Ti graphics processing units (GPUs). The code is available from [9].

2.2 Loss function

The loss function of a neural network is the objective that should be minimised in order to best describe the input data. A common simple loss function for a convolutional autoencoder is to take the mean squared error (MSE) between the input and output image, summing over all the pixels. Minimising the MSE means that the output of the network is as similar as possible to the input.

However, there are two problems with such a loss function. MSE is very susceptible to aliasing effects in which an output emission is in a neighbouring pixel to a similar input emission. The MSE penalises the network equally whether it produces an emission near to a target emission or very far away. The second problem is that the input event images are sparse; there are 4096 pixels in a 64×64 grid, but a single event may contain only $\mathcal{O}$ (10–100) emissions. Using a naive MSE loss means that the CNN will mainly learn about empty pixels, and will be biased towards producing no output activity.

JHEP12(2018)021

These two problems are solved by modifying the naive MSE.[1] Both the input target and the CNN output are blurred by using a set of truncated Gaussian kernels. A MSE-type loss is calculated for each Gaussian kernel, and a weighted sum of the losses is performed as in equation (2.3)

$$\begin{aligned} T^i_{\gamma\delta} &= \sum_{\mu\lambda} T_{\alpha\beta} G^i_{\alpha-\gamma,\beta-\delta} \\ O^i_{\gamma\delta} &= \sum_{\mu\lambda} O_{\alpha\beta} G^i_{\alpha-\gamma,\beta-\delta} \\ \mathcal{L} &= \frac{\sum_i w_i M\left(T^i_{\gamma\delta}, O^i_{\gamma\delta}\right)}{\sum_i w_i} \end{aligned} \tag{2.3}$$

where $T_{\alpha\beta}$ is the target event image that is input to the CNN and $O_{\alpha\beta}$ is the output image of the CNN. $G^i_{\gamma\delta}$ are a set of truncated Gaussian kernels and $T^i_{\gamma\delta}$ and $O^i_{\gamma\delta}$ are versions of the input and output, respectively, that have been blurred by $G^i_{\gamma\delta}$. The loss function, $\mathcal{L}$, is the weighted sum over an MSE-like function, $M\left(T^i_{\gamma\delta}, O^i_{\gamma\delta}\right)$ using weights w_i. The truncated Gaussian kernels are given by equation (2.4)

$$G^1 = \begin{bmatrix} 1 \end{bmatrix}, \qquad G^2 = \begin{bmatrix} 0.25 & 0.25 \\ 0.25 & 0.25 \end{bmatrix}, \qquad G^3 = \begin{bmatrix} 0.0947 & 0.118 & 0.0947 \\ 0.118 & 0.148 & 0.118 \\ 0.0947 & 0.118 & 0.0947 \end{bmatrix}. \tag{2.4}$$

The MSE-like function has two contributions, one from pixels in which there is activity in the target image, and one from pixels in which there is no target activity. The function $M\left(T^i, O^i\right)$ is given in equation (2.5)

$$M\left(T^i, O^i\right) = \frac{\sum_{\alpha,\beta}\left(T^i_{\alpha\beta} - O^i_{\alpha\beta}\right)^2 \mathcal{M}_{\alpha\beta}}{\sum_{\alpha,\beta}\mathcal{M}_{\alpha\beta}} + \frac{\left(\sum_{\alpha,\beta} O^i_{\alpha\beta}\left(1-\mathcal{M}_{\alpha\beta}\right)\right)^2}{\sum_{\alpha,\beta}\left(1-\mathcal{M}_{\alpha\beta}\right)} \tag{2.5}$$

where $\mathcal{M}_{\alpha\beta}$ is a mask image whose pixels have value 1 if the corresponding pixel in $T^i_{\alpha\beta}$ is non-zero, and are zero otherwise. The denominators in equation (2.5) account for the fact that events contain different numbers of target pixels with non-zero values. Without this normalisation by the number of active pixels there could be a bias towards fitting events that contain more target emissions. The normalisation ensures that the active and empty regions are given equal weight in the loss, while the overall effect of equation (2.5) is to treat all pixels that do not have any target activity in them as a single pixel.

[1] A theoretically nicer alternative might be to run a jet-finding algorithm, such as the k_t algorithm, on the output and the target, and compare infra-red safe jets. Using different jets with both large and small radius parameters would provide sensitivity to both wide- and small-angle emissions, down to a carefully chosen cut off. However, implementing the k_t algorithm on a GPU within TensorFlow is distinctly non-trivial, and offloading the image data from the GPU to the CPU for evaluation is computationally prohibitive.

JHEP12(2018)021

The weights, w_i, give some control over the degree to which the loss penalises the network for producing activity at a large angle to a target emission. Increasing weight w_3, which applies to the G^3 Gaussian kernel, causes the loss function to reduce the penalty for producing emissions at a wide angle to the target emission. Nevertheless, regardless of the weights chosen, the loss is always minimised by producing output in exactly the same pixel as the target. Apart from w_i, the custom loss does not introduce any additional model hyper parameters and the behaviour of the loss is determined by its functional form, which is chosen to stabilise the loss during training relative to a standard MSE.

2.3 Regularisation of network kernels

During training of the network, it may be possible for the convolutional kernel weights to diverge or become infinitesimal. In order to prevent this, dropout layers that randomly mask the output of the convolutional layers would typically be used to regularise the kernel weights. However, dropout cannot be used here because the max-pooling across convolutional filters is non-linear. Dropout works well when the network approximates a linear summation of neurons because it allows many subsets of the available network structures to be explored. However, in this model the convolutional kernels interact with each other, so it is not possible to drop a single network node without radically altering the network behaviour.

In lieu of dropout, a regularisation penalty is applied to prevent the learned convolutional kernel values from diverging. The kernels are similar to shower splittings, and so a regularisation term is added that penalises kernels that deviate from energy conservation. The kernel penalty term for each `Conv2D` filter with kernel weights $\vec{C}$ is given by equation (2.6)

$$\mathcal{R}\left(\vec{C}\right) = \lambda \times \left(1 - \sum_{i=1}^{k} C_i\right)^2 \tag{2.6}$$

where λ is a multiplier that controls the strength of the regularisation and the summation is over all of the kernel weights in the `Conv2D` layer. The penalty term for each kernel is calculated using equation (2.6) and added to the total loss for the model state during training.

The aim of the regularisation is to prevent the model reaching a state in which only a single filter is active during training so that interactions between filters can produce the desired complex behaviour. Since the `FilterMask` layer couples the activation rates of the `Conv2D` and `Conv2DTranspose` filters, it is sufficient to apply regularisation only to the `Conv2D` layers in order to prevent such convergence. In addition, if regularisation is applied to both the `Conv2D` and `Conv2DTranspose` filters then the space of possible filter configurations is greatly reduced and the model cannot achieve the desired complexity. Therefore regularisation is only applied to the `Conv2D` filters and is not applied to the `Conv2DTranspose` filters.

2.4 Model parameters

Having provided the general network structure, two sets of hyper-parameters define two concrete implementations of the CNN model. Model k_2 uses a kernel size of $k = 2$, and

JHEP12(2018)021

parameter	model k_2	model k_3
Kernel size, k	2	3
Input image size, N	64	81
Size of filter bank, F	9	7
Levels of decomposition	5	3
Regularisation, λ	500	300
Learning rate	5×10^{-5}	1×10^{-5}
Loss weight w_1	5	4
Loss weight w_2	2	2
Loss weight w_3	1	1
Total number of trained weights	72	126

Table 1. Model hyper-parameters.

model k_3 uses a kernel size of $k = 3$. The max-pooling and up-scaling used by the model requires that the input pixel array dimensions, $N \times N$, must obey the rule $N = k^n$, where n is an integer. Model k_2 is defined using input pixel arrays of size 64×64, and model k_3 uses inputs of size 81×81. The kernel and input array size together define the number of levels of convolution that the model performs; model k_2 is a narrower but deeper model with more levels of decomposition, while model k_3 uses a wider kernel but fewer levels of decomposition. The model hyper-parameters and other details are given in table 1

The choice of the size of the filter bank, F, is initially inspired by the desire for rotational invariance in the k_2 model. With $k = 2$ there are four possible rotations of the kernel, plus a parity transformation, meaning that eight filters can cover all possible transformations of one kernel. One additional filter is added in order to allow for a non-splitting. For the larger k_3 kernel, it is found that training a corresponding filter bank size of $F = 17$ is prohibitive, so the number of filters is reduced to $F = 7$.

The interaction between the filter kernels means that the behaviour of the model changes rather non-linearly with the size of the filter bank. If F is too small the model lacks the complexity to describe parton production, but if F is too large then the model complexity grows so much that training becomes very difficult. It is serendipitous that our initial choice of a filter bank size capable of providing exact rotational invariance in model k_2 is also about the right size to provide a viable model complexity. There remains sufficient flexibility in the model that the smaller filter bank in model k_3 is still capable of encoding the shower. Future improvements to the training procedure, either in the loss function or the optimisation algorithm, may permit a larger filter bank.

The other model hyper-parameters are decided by training models with different sets of parameter values using a small sample of the training events and inspecting the evolution of the training and validation losses. The size of the filter bank has by far the largest effect on the model, with the other parameters being of secondary importance. The regularisation λ can to some extent be used to control the complexity of the trained model. Lower λ

JHEP12(2018)021

values produce models that are easier to train but that are less complex and consequently produce less chaotic emissions, and generally fewer emissions overall. Values of λ in the range 100–1000 have been explored. The learning rate must be small enough that the model does not jump over the minimum in the loss function. The learning rates of table 1 are found to be sufficiently small; smaller values would also work, but would prolong the training time. The values of the loss weights, w_i, only enter the loss as a ratio because they are normalised by the sum of the weights. The choice of the weights values depends on the resolution of the images used with the model. Smaller pixels reduce the blur radius of the Gaussian filter, so the weight w_1 is slightly reduced for mode k_3, which has a slightly higher image resolution compared to mode k_2.

3 Training and evaluating the models

3.1 Monte Carlo event samples and selection

Simulated samples of proton-proton events are needed in order to train and to test the CNN models. Sherpa 2.2.4 [10–14] is used to generate a sample of 8.5 million QCD proton-proton collision events with up to four outgoing parton legs in the matrix element calculation. The default Sherpa tune is used, with the NNPDF 3.0 PDF set [15] and a shower merging scale of 20 GeV. The beam energy is 6.5 TeV per proton. Hadronisation and multi-parton interactions (MPI) are turned off because they have different scaling characteristics to a pure parton shower model, and would therefore require a more complicated deep learning model than is studied here.[2] The shower turn off scale is left at the Sherpa default value of 3 GeV. Sherpa's internal event selector is used to ensure that at least two R=0.4 jets with p_T greater than 25 GeV are produced by the matrix element calculation.

A subsequent event selection is made on the post-showering final-state particles that requires at least two R=0.4 anti-k_t jets [16] with p_T greater than 40 GeV and rapidity satisfying $|y| < (\pi - 0.4)$ in each event. The anti-k_t jet algorithm is run via the FastJet [17] library. This criterion selects approximately 0.5 million events from the initial 8.5 million generated events.

The CNN model requires pixel arrays as inputs. For this first implementation, a square pixel array using square pixels is used in order to avoid any unforeseen complications that might arise from using a image dimensionality that is not symmetric in rapidity-azimuth $(y - \phi)$. However, there is in principle no reason why this network design could not be extended to a larger rapidity range by using more pixels in the rapidity dimension. Each of the selected Sherpa events is converted to $N \times N$ pixel-array images in rapidity-azimuth by identifying the pixel in the $y - \phi$ plane into which each particle is emitted and adding the particle p_T to the pixel value. Pixels have a value of zero if no particles are emitted into them. The pixel array covers the rapidity range $-\pi \leq y < \pi$ and the

[2] Hadronisation is in any case a small-angle effect, and should not have a large contribution given that the events are converted into pixel arrays with finite pixel sizes. MPI will require further study to implement as part of a deep learning network, but given that MPI effects can be extracted by using a simple threshold in wavelet-space, it seems hopeful that much of the effect of MPI could be removed from training data by applying a threshold on network layer activity.

JHEP12(2018)021

azimuthal range $0 \leq \phi < 2\pi$. Each pixel array is normalised by dividing by the total sum of pixel values in the array and multiplying by $N \times N$ so that the average pixel value is one. The normalisation has no distorting effect on the event or the model, whose kernels learn about *ratios* between pixel values (which are unchanged by normalisation). However, normalisation is helpful in avoiding numerical issues that could otherwise potentially bias the model towards events with more activity. After normalisation, the arrays contain no information about the overall energy in the event, and the model can only learn about the shape of the radiation patterns.

Pixel arrays of size 64×64 and 81×81 are produced for use with models k_2 and k_3, respectively. Larger pixel arrays could be used, but due to the requirement that $N = k^n$, they would need to be a minimum of 128×128 and 243×243, which makes training the model considerably more difficult. Studies of the angular separation between individual particles produced by Sherpa's shower show that pixel sizes around 0.1×0.1 have sufficient resolution to capture the vast majority of the details of the radiation patterns. Pixel arrays are therefore limited to 64×64 and 81×81.

In order to test the trained CNN shower models, unshowered partons produced by a matrix element calculation are needed. An additional sample of 8.5 million matrix element (ME) Sherpa events generated without any parton shower is used for this purpose. These events have the same generator-level process and selection of two R=0.4 jets with p_T greater than 25 GeV, but lack any post-shower selection. The full sample of 8.5 million ME events is converted to $N \times N$ pixel arrays, but is not normalised.

A shower merging scheme for the CNN shower models is introduced in section 3.4. As a test of this scheme, two further Sherpa samples of 8.5 million events — one with showering, one without — are produced with an alternative merging scale of 40 GeV (compared to 20 GeV for the nominal samples). These alternative samples are not used for training and are only used to check that the results of the CNN merging scheme are not dependent on the merging scale.

3.2 Training on showered events

The fully showered event pixel arrays are divided into a training sample, which contains 90% of the 0.5 million images, and a validation sample, which contains the remaining 10%.

Models are trained for several hundred epochs using the learning rates given in table 1. The Nadam optimiser [18, 19] is used and the network weights are initialised with random values from the Glorot Normal distribution [20]. An example of the evolution of the training and validation loss for model k_2 is shown in figure 3a. During training, the model rests for some number of epochs in a moderate loss state, before falling into a lower loss state. After spending a small number of epochs in the low loss state, the model then undergoes a rapid increase in both training and validation loss and reaches a high loss state, before falling once again to a (different) moderate loss state. This evolution of both training and validation loss is shown in figure 3. Note the lack of divergence between the training and validation loss, indicating that the rapid increase is not due to overfitting the training data. The lack of divergence is seen regardless of the choice of model hyper-parameters (section 2.4). This lack of over-fitting is probably due to the high dimensionality of the

JHEP12(2018)021

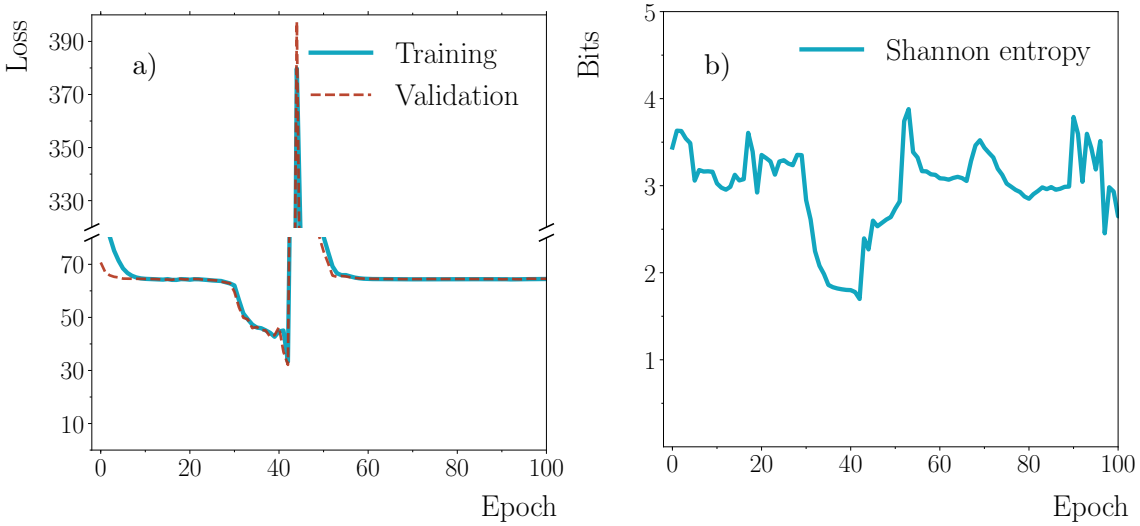


Figure 3. Evolution of the model during training. a), left, shows how the loss function evaluated on the training and validation event samples changes during training, while b), right, shows the evolution of the Shannon entropy of the `FilterMask` layer.

input data together with the relatively high statistical power of the input and the rather small number of learned parameters. The lack of divergence is also a good indicator that the training set of events is (more than) sufficient to reach the maximum potential performance of the trained model.

The probabilities stored in the `FilterMask` layers are the rates with which each filter is active, and the Shannon entropy[3] of those probabilities is a measure of the complexity of the model state. If the Shannon entropy is high, the filters are all active with similar rates, whereas if the entropy is low then a small number of the filters dominate the description of the collision events. Figure 3b shows the evolution of the Shannon entropy of the model during the same training period as figure 3a. The entropy is normalised so that each `FilterMask` layer encodes at most one bit of information, and for $k = 2$ there are five `FilterMask` layers, so there is a maximum of five bits stored in the model. At the start of training, the model is in a high entropy state because the filters are active with quasi-random probabilities. When, during training, the loss falls to a minima, the entropy also declines to moderate values, showing that the model is in a more ordered state. When the loss subsequently rises rapidly, the model undergoes a transition to a higher entropy state again. When trained for a very large number of epochs (not shown here), the entropy of the model gradually evolves to a very low entropy state, while the loss remains on a plateau. This long-term decline in entropy accompanied by a near-constant loss indicates that there are a large number of model states that are equally good at describing the collision events, and the model tends towards a state in which the model behaviour is dominated by a small number of the filters in the filter bank. This behaviour of the model complexity could in future potentially be used in more effective network training algorithms.

[3]The Shannon entropy, H, of a single `FilterMask` layer is given by $H = \frac{\sum_{i=0}^{F} p_i \ln p_i}{\ln F}$, where p_i are the probabilities stored in the layer and F is the size of the filter bank.

JHEP12(2018)021

Since the loss reaches a plateau after many training epochs, while the model complexity continues to reduce, the optimal model configuration is selected as the epoch that corresponds to the minimum loss prior to the rebound onto the plateau. In the example of figure 3 this corresponds to around the forty second epoch. A callback function is used within Keras to save the corresponding best model state.

3.3 Comparison with class IV Cellular Automata

Parton showers and the CNN implemented here are similar in both conception and behaviour to Cellular Automata (CA). Cellular Automata evolve a system from an initial state using a set of rules that describe how the current state should change under a discrete step. Parton showers employ a set of splitting rules to evolve the state of the shower between scales. Similarly, the CNN uses rules described by the `Conv2D` or `Conv2DTranspose` layers to step between angular scales. The change in the CNN during training shares some interesting aspects with the change in behaviour of CA as they move through their available "rule space".

Cellular automata have been divided into four classifications [21–23]:

- Class I: the initial state evolves to a fixed pattern, and is not interesting for the present study
- Class II: the evolution from the initial state is dominated by well-ordered periodic structures that are largely independent of the initial state.
- Class III: the system evolves chaotically and produces random patterns that are independent of the initial state.
- Class IV: the system evolves to produce complex states that are neither completely chaotic, nor completely ordered. The evolution is dependent on the initial state, and the rules interact in a non-linear way to produce complex behaviour.

One key indicator of the difference between classes II, III and IV is the entropy of the CA site activity. Class II typically produces low entropy states, while Class III produces high entropy states. As the rules of the CA are altered, it can undergo a (potentially rapid) phase transition between the classes.

Prior to training,[4] the CNN behaves very much like a Class III Cellular Automata. The network weights are initialised to random values and produce a large number of random emissions, uncorrelated with the input to the network.

After over-training[5] for many hundreds of epochs, the network behaves like a Class II Cellular Automata. The Shannon entropy of the `FilterMask` layers declines, indicating

[4]The CNN should always be trained for one epoch on a small number of events to ensure that the probabilities in the `FilterMask` layers have been calculated correctly. If the network is not trained at all, the probabilities will violate unitarity. As long as training is sufficiently short, the network kernels will remain random.

[5]Over-training here means the model continues training long after the loss function has been minimised so that the Shannon entropy declines.

JHEP12(2018)021

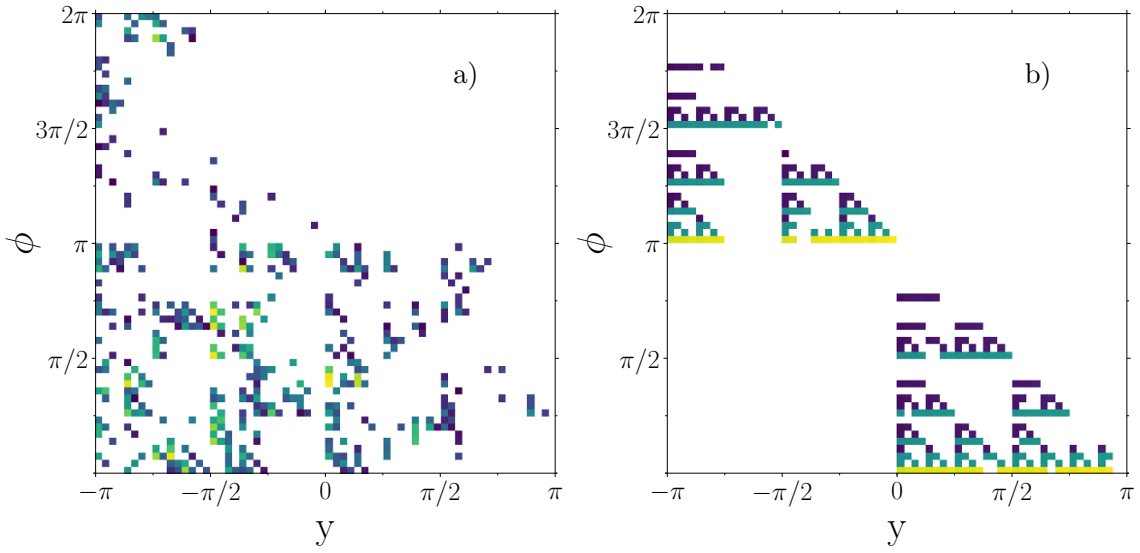


Figure 4. Example emission patterns produced by an untrained (chaotic) CNN (a) and an over-trained (periodic) CNN (b). The input to the CNN was the same in both cases.

that the model is highly-ordered and a small number of kernels dominate the evolution of the input through the network.

The evolution of the CNN from chaotic to highly ordered behaviour is illustrated by some radiation patterns from model k_2 in figure 4. Figure 4a is the output of the untrained model k_2 when two random partons are used to seed the CNN. The output in figure 4a is chaotic, there is no pattern and the output is uncorrelated with the input. Figure 4b shows the output from the same model k_2 using the same input when the model is over-trained by several hundred epochs. The model output has become sparse, with well-ordered structures that are repeated over different angular scales.

As the rules of the CA are updated so that a transition from class II to class III is made, there can — depending on the specifics of the CA — be a Goldilocks region in which the CA is class IV and is capable of describing complex phenomena. Similarly, as the CNN kernels are updated during training, it becomes capable of describing the complex behaviour of a parton shower close to the transition from chaos to periodicity. The goal for training a CNN capable of behaving as a parton shower should thus be to maximise the time spent exploring the kernel parameters in the transition region.

3.4 Merging CNN with matrix element calculations

The trained network is evaluated on the pixel arrays produced from Sherpa matrix element events that have not previously been showered. The effect of the `FilterMask` layer in this case is to randomly activate filters according to the rate with which they were active during training. The output of the evaluation on un-showered events is an approximation to fully showered events. The output of the CNN is a pixel array of the same size as the input. These output arrays are converted back into lists of particle-level collision events by creating a single particle for each pixel that has a p_T value above 100 MeV. The particle

JHEP12(2018)021

p_T is the same as the pixel p_T, and the particle is emitted into a random location within the pixel.

Care must be taken to merge parton showers with perturbative matrix element calculations in order to prevent the double-counting of emissions from both the shower and the ME into the same region of phase space. The Sherpa matrix elements used here assume a k_T ordered parton shower, where k_T is the transverse momentum of an emitted parton relative to the emitter. Although the CNN does not currently explicitly define an ordering parameter, it most closely resembles an angular ordering. This mis-match between the ordering used in the ME and the CNN implies that a veto should be applied to the CNN to prevent emission of shower partons into phase-space regions that should be covered by the ME [24]. This shower merging is implemented as a new layer in the CNN that is added after each `Conv2DTranspose` level. The shower merging is only used during evaluation of the CNN on pixel arrays produced from matrix element events. The shower merging is *not* used during training on fully showered events.

Each application of the `Conv2DTranspose` filters corresponds to the generation of potential new parton emissions, which is why the shower merging veto is applied after each new application of the `FilterMask` and `Conv2DTranspose` layers. The shower veto scale evolves with depth in the CNN. The network bottleneck in the middle of the autoencoder corresponds to the widest angle emissions, and is analogous to the shower starting scale. The shower veto scale at the bottleneck is therefore determined from that provided by Sherpa's matrix element, Q_0, in this case $Q_0 = 20\,\text{GeV}$, in combination with the emission angle to which the bottleneck corresponds. The emission angle, $\Delta\phi$, at the bottleneck is approximately given by $\Delta\phi \simeq \pi/k$, and the shower veto scale, Q, at the bottleneck is therefore approximated by equation (3.1)

$$Q\left(\Delta\phi\right) = Q_0/\left(1 - \cos\Delta\phi\right) . \tag{3.1}$$

Each de-convolutional level within the network corresponds to a different emission angle, and for each step in network level away from the bottleneck, the emission angle is divided by a factor of k. The shower veto scale, Q, in a given network layer is thus determined from the running emission angle together with equation (3.1) Thus the later layers of the re-inflation stage, which correspond to smaller angles, use a larger veto, which is appropriate as it allows collinear emissions more easily than wide angle emissions.

At each level of the re-inflation, the veto procedure is applied as follows:

- The output of the merged `Conv2DTranspose` bank is sub-divided into windows the same size as the $k \times k$ convolutional kernel.
- Any pixel below the veto scale for that convolutional level is left unchanged.
- The number of pixels, N_S, in each window that are above the veto scale is calculated.
- The number of pixels, N_{ME}, in each window of the corresponding layer on the compression side of the CNN is calculated.

JHEP12(2018)021

- If $N_{\mathrm{ME}} \geq N_S$ then all pixels in the output window are accepted because no new emissions above the merging scale have been added in that region.
- If $N_{\mathrm{ME}} < N_S$ then the pixels above the merging scale in that window are replaced by those in the corresponding window of the image from the compression-side of the CNN. These pixels correspond to the state of the shower prior to splitting.
- Replacement hard pixels are adjusted to account for the emission of any soft pixels below the merging scale within the same window.

This merging procedure can be carried out entirely via matrix manipulation operations, and is implemented as a Keras layer and executed on the GPU.

4 Jet distributions predicted by the CNN

The Rivet analysis framework [25] is used to compare events showered with Sherpa to events showered with the CNN, as well as ME-level events that have not been showered. A sample of eight million events produced with Herwig 7.1.4 [26–30] is also used as an example of an angular ordered shower for comparison with Sherpa's k_T ordered shower. For these events, the default Herwig tune is used with a beam energy of 6500 GeV per proton. The QCD $2 \to 2$ process is used and, for a direct comparison with the Sherpa samples, both hadronisation and MPI are turned off.

Analyses are performed using two different jet algorithms from the FastJet package: the anti-k_t algorithm with a radius parameter of 0.4, and the k_t algorithm [31] with a radius of 0.6. Jets must have a rapidity that satisfies $|y| < 2.5$ and transverse momentum that satisfies $p_T > 40\,\mathrm{GeV}$.

Some example emission patterns generated by the CNN models k_2 and k_3 are shown in figure 5 for events that satisfy the anti-k_t R=0.4 jet selection. The ME partons that are used to seed the CNN are shown as grey crosses. The CNN splits these partons into a shower of lower energy partons in the region of the initial parton. The CNN is also able to generate some wider-angle activity; for example, in the middle panel, model k_2 has generated a jet around $\{y, \phi\} = \{-\pi/2, \pi/4\}$.

The distribution of the number of jets in each event that satisfy the jet requirements are shown in figure 6. The matrix element can (rarely) generate at most four partons, so events that contain more than four jets have had those jets generated by the parton shower. Model k_2 produces somewhat fewer high multiplicity events than the target Sherpa model, while model k_3 produces slightly too many high multiplicity events. It is encouraging that the two CNN models bracket the target data, which suggests that a CNN could indeed be made to describe the jet multiplicity after further adjustment and improvement.

The jet width,[6] ρ, is a test of the shape of the radiation pattern emitted around a jet, and is shown in figure 7 for all jets that satisfy the selection criteria. The simple CNN

[6] ρ is given by $\rho = \frac{\sum_i \Delta R(j,p^i) p_T^i}{\sum_i p_T^i}$ where the sum is over all constituents of the jet, p_T^i is the p_T of the i^{th} jet constituent and $\Delta R\,(j, p_i)$ is the angular separation between that constituent and the jet axis.

JHEP12(2018)021

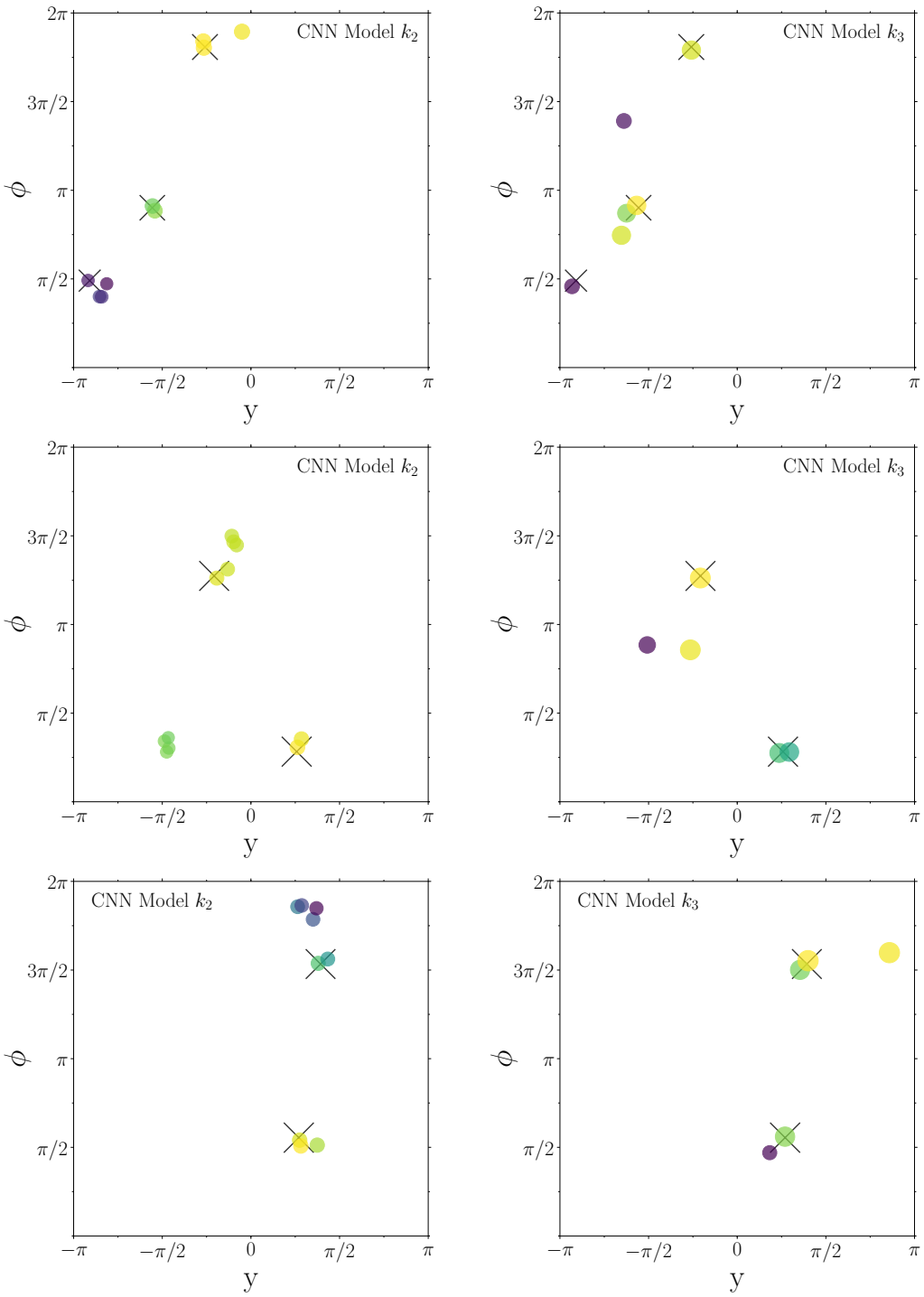


Figure 5. Three example emission patterns generated by the k_2 (left column) and k_3 (right column) CNN models. The coloured circles indicate the location of emitted particles, and the sizes and colours of the circles indicate the particle p_T. The grey Xs in each plot are the matrix element emissions that are input to the CNN.

JHEP12(2018)021

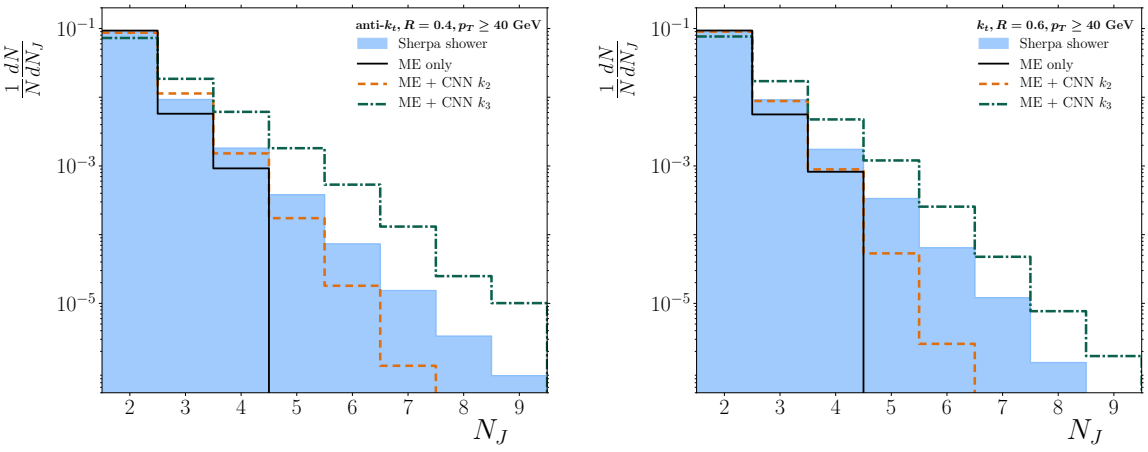


Figure 6. Number of jets per event using the anti-k_t R=0.4 jet algorithm (left) and the k_t R=0.6 algorithm (right).

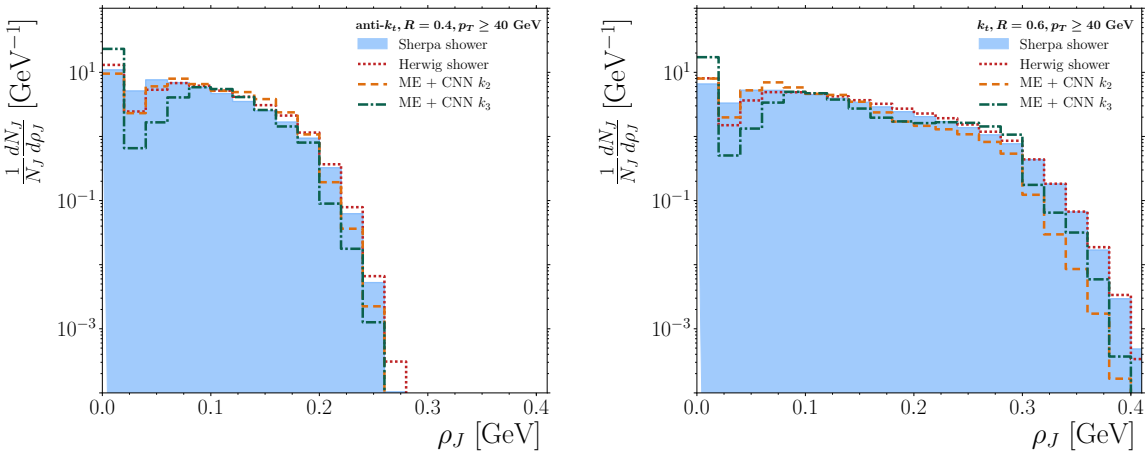


Figure 7. Jet width distributions using the anti-k_t R=0.4 jet algorithm (left) and the k_t R=0.6 algorithm (right).

models do a surprisingly good job of recreating the jet shapes of the true parton shower, especially for the large width jets. There is a deficiency in small width jets compared to Sherpa, and an over-abundance of zero-width jets. This suggests some kind of dead cone effect, which could be an artefact of the approximate merging procedure, or some other effect of using an angular ordered-type shower. By way of comparison, Herwig's angular-ordered shower also displays a similar dip in the number of low width jets and shows the range of expected differences between an angular-ordered shower and a k_T ordered shower. The CNN models have no information about parton mass, and also have a cut off at small angle due to the finite pixel size, both of which may affect the small width jets to some extent.

Jet masses arise from the finite width of the jet, and jet mass distributions also serve as a test of the radiation emitted around a jet. The distributions of jet masses from all selected jets are shown in figure 8. Both the k_2 and k_3 CNN models have generated smooth

JHEP12(2018)021

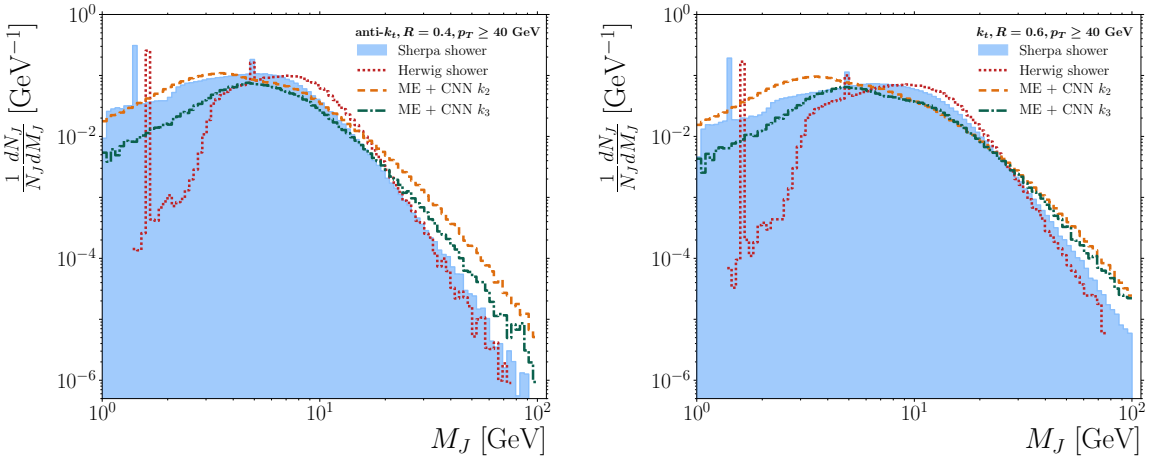


Figure 8. Jet mass distributions using the anti-k_t R=0.4 jet algorithm (left) and the k_t R=0.6 algorithm (right).

mass distributions from the input ME partons, with gradients close to those of the target Sherpa model in the tails. However, the peak of the mass distributions do not match the target. This is not surprising because the CNN models do not contain any information about mass and do not trace the parton masses through the network; jet masses arise only from the angular width of the jets. Furthermore, the existence of massive b and c quarks can be seen in the Sherpa mass distribution as the small spikes at around 4.5 and 1.7 GeV, respectively. Since the CNN does not include any mass term for the partons (or pixels) it cannot reproduce these spikes. Again, the Herwig shower is shown as an example of the differences that can be expected between angular and k_T ordered showers, in particular in the low mass region.

Finally, the transverse momentum (p_T) distributions of all jets that satisfy the selection criteria are shown in figure 9. Both of the CNN models improve the jet p_T spectra relative to the unshowered matrix element partons by increasing the proportion of high-p_T jets and flattening the bump[7] in the ME distribution between 40 and 50 GeV. Model k_3 is very close to the p_T spectrum of the target Sherpa parton shower for both jet algorithms. However, model k_2 is somewhat too hard, and shows a flattening of the spectrum around 80 GeV. This flattening is an artefact of the shower merging procedure and disappears if the merging layer is removed from the CNN.

As a test of the shower merging procedure described in section 3.4, distributions for the number of jets and the jet p_T using a second set of matrix element events with a higher merging scale of $Q_{\text{cut}} = 40$ GeV are produced. The shower veto scale in the network merging layer is also updated from 20 GeV to 40 GeV to match the input matrix element calculation, and the two CNN models with this updated scale are used to shower the matrix elements. A comparison between the results from the $Q_{\text{cut}} = 20$ GeV and the $Q_{\text{cut}} = 40$ GeV samples for the jet multiplicity and jet p_T distributions are shown in

[7]This small bump occurs because the ME event selection requires the sub-leading jet to pass the same $p_T > 40$ GeV criterion as the leading jet.

JHEP12(2018)021

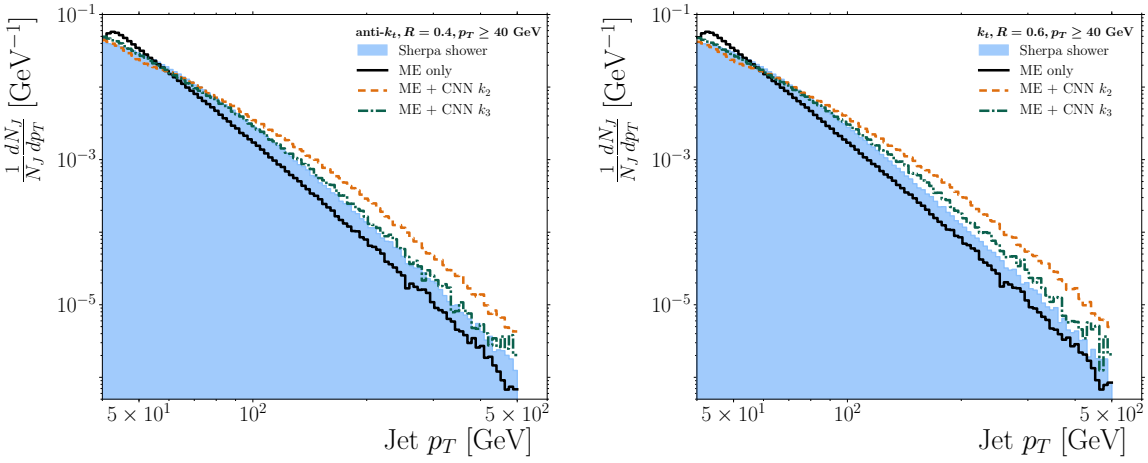


Figure 9. Jet p_T distributions using the anti-k_t R=0.4 jet algorithm (left) and the k_t R=0.6 algorithm (right).

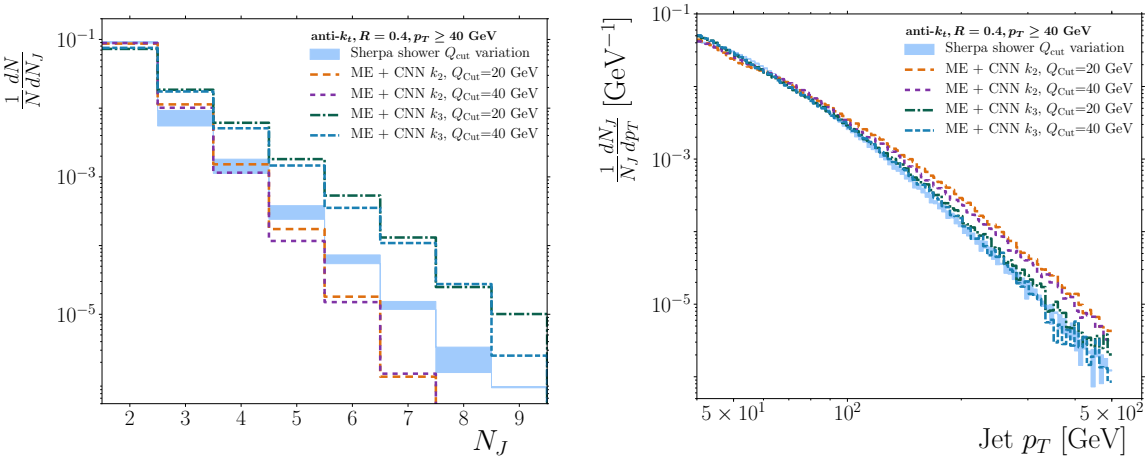


Figure 10. Jet multiplicity (left) and p_T (right) distributions compared for merging scales of 20 GeV and 40 GeV for models k_2 and k_3. The merging scale variation using Sherpa's native shower with merging scales of 20 and 40 GeV is shown as the shaded blue band.

figure 10. While there is a small difference between the results with the two merging scales, this is consistent with the difference when using the two different merging scales with Sherpa's native shower, as shown by the shaded blue band in figure 10. That the consistency of the CNN merging scheme at different merging scales is comparable to that of Sherpa's own shower is somewhat surprising given that the discrete angular scales used by the CNN imply an inherent ambiguity in the evolution of the merging veto scale with network depth.

5 Features learned by the CNN

Each level of decomposition within the model network corresponds to a different angular scale for emissions. Deeper layers — those closer to the compression bottleneck — capture

JHEP12(2018)021

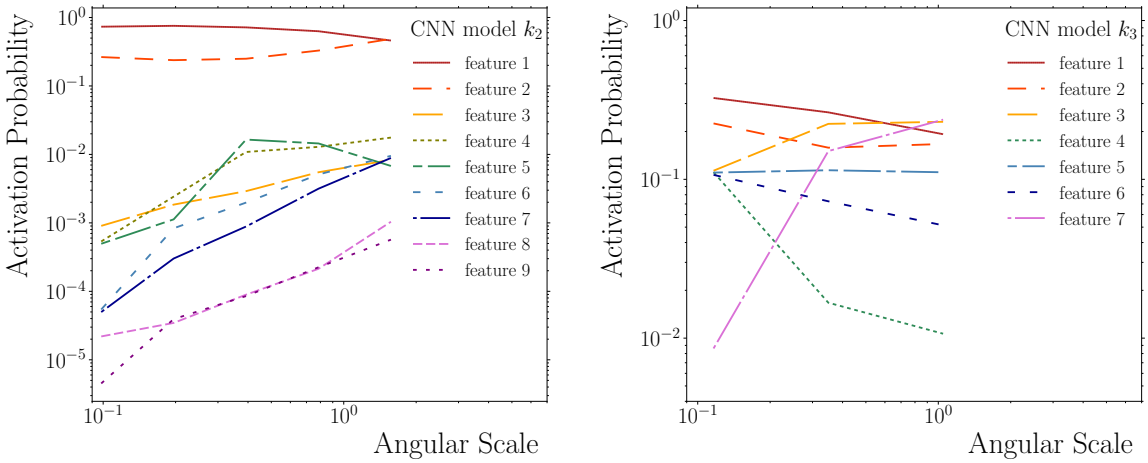


Figure 11. Evolution of the feature activation rates with angular scale for model k_2 (left) and model k_3 (right).

larger angles, while layers at the very top and bottom of the network have a smaller receptive field and reproduce small angle effects. Each level of decomposition has its own `FilterMask` layer, with each `FilterMask` storing a different set of probabilities for the filter activations at the different network depths. Thus by converting the decomposition level — given by the network depth — to an angular scale, it is possible to study the evolution of the filter activation rates with angular scale. The angular scale, $\Delta\phi$, to which the l^{th} level of the decomposition of a $N \times N$ image corresponds to is given by equation (5.1)

$$\Delta\phi = \pi \frac{k^l}{N} \,. \tag{5.1}$$

Figure 11 shows the evolution of the filter activation rates with the angular scale. The set of F individual filters are labelled feature 1-F, where F is the number of filters in the model and feature 1 is defined as the filter with the largest activation probability at small angles, while feature F is the filter with the smallest activation probability at small angle. Recall that the same set of filters are used at all angular scales, it is only the filter activation probability that changes with angular scale. The activation probabilities exhibit some interesting behaviour that is suggestive of interactions between the different filters. For example, model k_2 appears to show bands of filters having similar activation rates, while model k_3 also shows apparent correlations in the activation rates at different scales.

The features that each filter in the model encode are revealed by fixing the model weights so that only a single pair of `Conv2D` and `Conv2DTranspose` filters is active. A randomly generated pixel array is input to this sub-model and then updated using the output of that model. The update is repeated twenty times until a stable pixel array is converged upon. This iterative procedure is itself repeated using one hundred different random starting arrays, and the average is taken of the results.

The features encoded by the nine filters in model k_2 are shown in figure 12, and the seven features encoded by the filters of model k_3 are shown in figure 13. All the features exhibit self-similarity, which is a result of the use of the same filter at multiple angular

JHEP12(2018)021

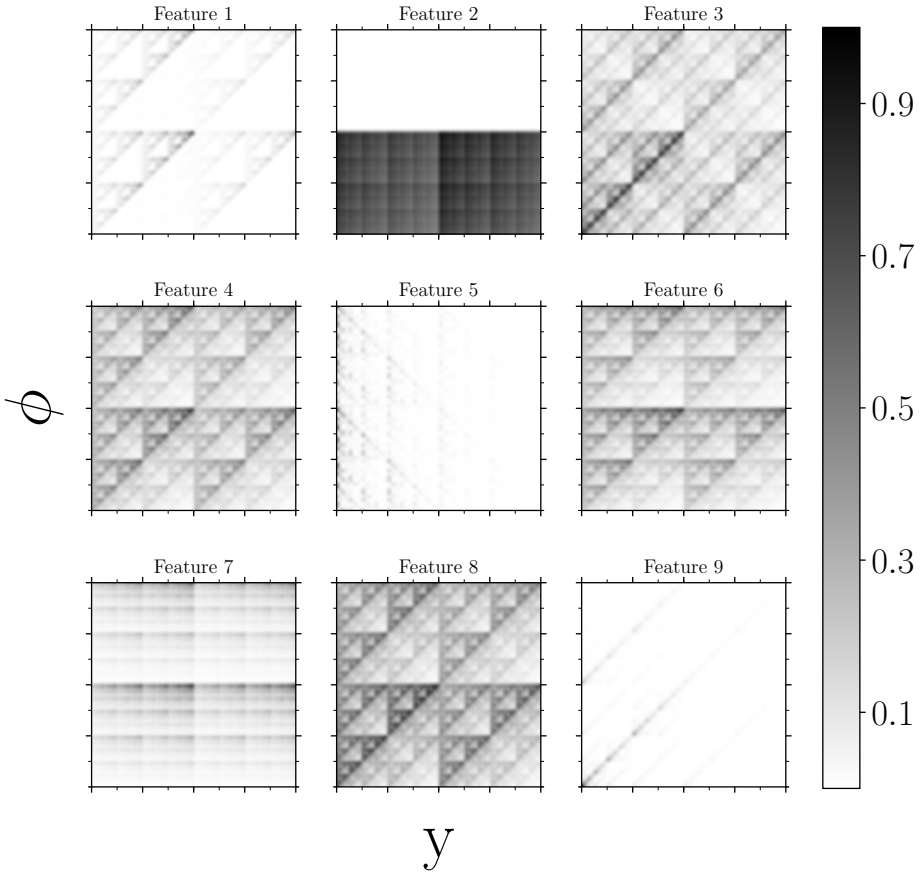


Figure 12. Features encoded in model k_2.

scales. The features encoded in model k_3 are more complex than those in model k_2 due to the larger kernel size of the former.

6 Concluding remarks

We have demonstrated that it is possible to encapsulate many of the features of a QCD parton shower in an autoencoding recursive convolutional neural-network, and that the number of trainable network parameters needed to do so is not large. The network design is inspired by self-similarity and wavelet decomposition, which significantly reduces the number of network parameters.

The CNN learns features from jet events without needing any human supervision to classify the events or objects within them. Convolutional neural networks have often been used as a tool for event or jet classification [32, 33], but here we have shown that they may also be used to learn features from QCD directly, with minimal assumptions about jet behaviour and no requirement that jets be classified using human-defined labels. Using concepts like quark/gluon tagging as a target for neural networks in supervised learning imposes human (mis-) concepts and biases onto the data, but the basis set of features learned during the course of training a jet tagger is in many ways more interesting than

JHEP12(2018)021

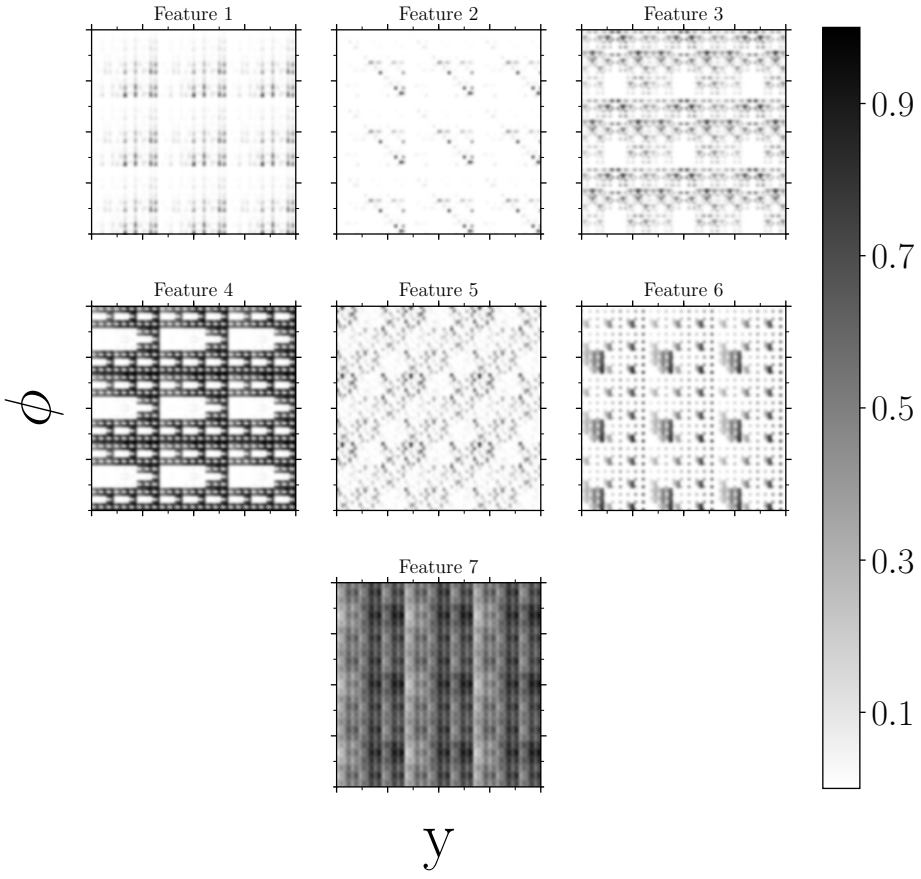


Figure 13. Features encoded in model k_3.

the tagger itself. Here we have attempted to ask the more general question "what features may be learned from QCD while imposing as limited a set of biases as possible?"

The generative model described in [6] has a similar goal to our present work of using CNNs to learn features from jets. However, the adversarial approach used there is quite different to our autoencoding model. Among other things the adversarial model lacks the explicit recursion that limits the number of model parameters. Recursively using convolutional kernels as analogues of shower splitting functions and interpreting depth within the network as the splitting angle allows the autoencoding CNN to be merged with a fixed-order matrix element to shower entire collision events. On the other hand, the adversarial model can only generate single jet images. There may even be the potential to combine the two approaches in the same way that traditional event generators combine separate models for showering, MPI and hadronisation. The output of an autoencoding shower model could be passed to a further generative model to add non-perturbative details that are missing from the simpler — but more easily interpreted — recursive shower picture.

Recursive (but not convolutional) networks that use particle four-vectors as inputs have also been used both to classify jets [34] and to learn particle probability distributions that can form the basis of a generative model in JUNIPR [5]. The JUNIPR model takes its hierarchical ordering from a jet clustering algorithm, and although that model has enough

JHEP12(2018)021

flexibility to work with any clustering hierarchy, the performance does show some residual dependance on the choice of jet algorithm. In contrast, an angular (but not jet-derived) hierarchy is built into the structure of our autoencoding model. JUNIPR and our model use similar ideas to capture features of QCD, though expressed in slightly different ways with slightly different aims. Having shown that recursion can be directly included in image-based models, it may be that vector-based and image-based models evolve towards the same sort of model structure. Indeed, recent work has shown that convolutional models may soon be used for tasks for which recurrent or recursive models have traditionally been used, such as language translation [35].

Convolutional networks using images have sometimes been criticised as a tool for QCD because they use a large number of learned network weights. We have shown that a large number of learned CNN parameters is not necessary to describe QCD, and that self-similarity can be implemented with recursion in image-based models.

This method also bears some comparison with shower deconstruction [36]. In shower deconstruction, a simplified parton shower model is used to estimate the probability that a given parton configuration originates from either signal or background. The probability is estimated by evaluating the history of the splitting terms that lead to the final parton configuration. Similarly, the compression stage of the autoencoder matches a series of learned kernels to the input parton configuration, taking the best match at each stage via the max-pooling operation. In this way, the autoencoder tests, in parallel, a very large number of possible shower histories. Similarly to shower deconstruction, an autoencoder trained on QCD background data could be used to discriminate between background and signal because the non-QCD signal will not match the learned shower behaviour.

Though far from perfect, the simplified models used here are able to capture qualitatively the behaviour of the Sherpa target on which they are trained. The models could equally have been trained directly on appropriate jet data from the Large Hadron Collider, with the proviso that some important non-perturbative effects (hadronisation and MPI) were (intentionally) left out here. This raises the interesting possibility that, with further improvement — particularly by adding a mass term — deep neural networks can provide a new way of learning about QCD that allow for data-driven background models that do not use any assumptions beyond the basic network structure.

Acknowledgments

Thanks to Alejandro Alonso for helping to configure a pair of GPUs for TensorFlow, and for technical suggestions related to Keras. Thanks also to Frank Krauss for a useful discussion and suggestions about shower merging and shower deconstruction. Finally, thanks to Peter Hansen, Troels Petersen and Stefania Xella, who provided useful feedback that improved the paper. This work was funded by the Danish National Research Foundation.

Open Access. This article is distributed under the terms of the Creative Commons Attribution License (CC-BY 4.0), which permits any use, distribution and reproduction in any medium, provided the original author(s) and source are credited.

JHEP12(2018)021

References

[1] J.W. Monk, *Wavelet Analysis: Event De-noising, Shower Evolution and Jet Substructure Without Jets*, arXiv:1405.5008 [INSPIRE].

[2] P. Mehta and D.J. Schwab, *An exact mapping between the Variational Renormalization Group and Deep Learning*, arXiv:1410.3831.

[3] C. Bény, *Deep learning and the renormalization group*, arXiv:1301.3124.

[4] D. Oprisa and P. Toth, *Criticality & Deep Learning II: Momentum Renormalisation Group*, arXiv:1705.11023.

[5] A. Andreassen, I. Feige, C. Frye and M.D. Schwartz, *JUNIPR: a Framework for Unsupervised Machine Learning in Particle Physics*, arXiv:1804.09720 [INSPIRE].

[6] L. de Oliveira, M. Paganini and B. Nachman, *Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis*, *Comput. Softw. Big Sci.* **1** (2017) 4 [arXiv:1701.05927] [INSPIRE].

[7] F. Chollet et al., *Keras*, (2015) https://keras.io.

[8] M. Abadi et al., *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*, arXiv:1603.04467 [INSPIRE].

[9] J. Monk, *APEMEN. Autoencoding Parton Emitting Model Encoded in Networks*, (2018) https://github.com/twoev/APEMEN.

[10] T. Gleisberg et al., *Event generation with SHERPA 1.1*, *JHEP* **02** (2009) 007 [arXiv:0811.4622] [INSPIRE].

[11] S. Schumann and F. Krauss, *A Parton shower algorithm based on Catani-Seymour dipole factorisation*, *JHEP* **03** (2008) 038 [arXiv:0709.1027] [INSPIRE].

[12] F. Krauss, R. Kuhn and G. Soff, *AMEGIC++ 1.0: A Matrix element generator in C++*, *JHEP* **02** (2002) 044 [hep-ph/0109036] [INSPIRE].

[13] T. Gleisberg and S. Hoeche, *Comix, a new matrix element generator*, *JHEP* **12** (2008) 039 [arXiv:0808.3674] [INSPIRE].

[14] S. Hoeche, F. Krauss, S. Schumann and F. Siegert, *QCD matrix elements and truncated showers*, *JHEP* **05** (2009) 053 [arXiv:0903.1219] [INSPIRE].

[15] NNPDF collaboration, R.D. Ball et al., *Parton distributions for the LHC Run II*, *JHEP* **04** (2015) 040 [arXiv:1410.8849] [INSPIRE].

[16] M. Cacciari, G.P. Salam and G. Soyez, *The anti-k_t jet clustering algorithm*, *JHEP* **04** (2008) 063 [arXiv:0802.1189] [INSPIRE].

[17] M. Cacciari, G.P. Salam and G. Soyez, *FastJet User Manual*, *Eur. Phys. J.* **C 72** (2012) 1896 [arXiv:1111.6097] [INSPIRE].

[18] T. Dozat, *Incorporating Nesterov Momentum into Adam*, in proceedings of the *International Conference on Learning Representations 2016 (ICLR 2016)*, Caribe Hilton, San Juan, Puerto Rico, 2–4 May 2016.

[19] S. Ruder, *An overview of gradient descent optimization algorithms*, arXiv:1609.04747 [INSPIRE].

JHEP12(2018)021

[20] X. Glorot and Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*, in proceedings of *Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010, volume 9, Y.W. Teh and M. Titterington eds., PMLR (2010), pp. 249–256.

[21] S. Wolfram, *Universality and complexity in cellular automata*, *Physica D* **10** (1984) 1.

[22] W. Li, N.H. Packard and C.G. Langton, *Transition phenomena in cellular automata rule space*, *Physica D* **45** (1990) 77.

[23] G.J. Martinez, J.C. Seck-Tuoh-Mora and H. Zenil, *Computation and Universality: Class IV versus Class III Cellular Automata*, *J. Cell. Automata* **7** (2013) 393 [arXiv:1304.1242].

[24] K. Hamilton, P. Richardson and J. Tully, *A Modified CKKW matrix element merging approach to angular-ordered parton showers*, *JHEP* **11** (2009) 038 [arXiv:0905.3072] [INSPIRE].

[25] A. Buckley et al., *Rivet user manual*, *Comput. Phys. Commun.* **184** (2013) 2803 [arXiv:1003.0694] [INSPIRE].

[26] G. Marchesini and B.R. Webber, *Simulation of QCD Jets Including Soft Gluon Interference*, *Nucl. Phys.* **B 238** (1984) 1 [INSPIRE].

[27] G. Marchesini and B.R. Webber, *Monte Carlo Simulation of General Hard Processes with Coherent QCD Radiation*, *Nucl. Phys.* **B 310** (1988) 461 [INSPIRE].

[28] S. Gieseke, P. Stephens and B. Webber, *New formalism for QCD parton showers*, *JHEP* **12** (2003) 045 [hep-ph/0310083] [INSPIRE].

[29] M. Bahr et al., *HERWIG++ Physics and Manual*, *Eur. Phys. J.* **C 58** (2008) 639 [arXiv:0803.0883] [INSPIRE].

[30] L. Lönnblad, *Development strategies for PYTHIA version 7*, *Comput. Phys. Commun.* **118** (1999) 213 [hep-ph/9810208] [INSPIRE].

[31] S.D. Ellis and D.E. Soper, *Successive combination jet algorithm for hadron collisions*, *Phys. Rev.* **D 48** (1993) 3160 [hep-ph/9305266] [INSPIRE].

[32] L. de Oliveira, M. Kagan, L. Mackey, B. Nachman and A. Schwartzman, *Jet-images — deep learning edition*, *JHEP* **07** (2016) 069 [arXiv:1511.05190] [INSPIRE].

[33] P.T. Komiske, E.M. Metodiev and M.D. Schwartz, *Deep learning in color: towards automated quark/gluon jet discrimination*, *JHEP* **01** (2017) 110 [arXiv:1612.01551] [INSPIRE].

[34] G. Louppe, K. Cho, C. Becot and K. Cranmer, *QCD-Aware Recursive Neural Networks for Jet Physics*, arXiv:1702.00748 [INSPIRE].

[35] M. Elbayad, L. Besacier and J. Verbeek, *Pervasive Attention: 2D Convolutional Neural Networks for Sequence-to-Sequence Prediction*, arXiv:1808.03867.

[36] D.E. Soper and M. Spannowsky, *Finding top quarks with shower deconstruction*, *Phys. Rev.* **D 87** (2013) 054012 [arXiv:1211.3140] [INSPIRE].

JHEP12(2018)021